HTMLオブジェクト

- HTMLタグをドリトル内で直接扱うことができるオブジェクトです。
- 角括弧（<>）を使って直接記述することでHTMLオブジェクトを作成できます。
- HTMLオブジェクトを使うことで、動的な画面更新が可能な双方向コンテンツを簡潔に記述できます。
- WebAPIオブジェクトと組み合わせることで、サーバとの通信を行う双方向コンテンツを作成できます。

HTMLオブジェクトの基本

- HTMLタグは数値や文字列と同じように、角括弧（<>）で囲んで作成します。
    - （例）順序なしリスト(ul)タグを作成します。

```
メッセージボード←<ul></ul>
```

- （例）入力欄(input)タグを作成します。

```
入力欄←<input>
```

- （例）ボタン(button)タグを作成します。

```
送信ボタン←<button>送信</button>
```

- HTMLオブジェクトには、それぞれの要素に応じた様々なメソッドが用意されています。
- 生成されたHTMLタグはオブジェクトであり、プロパティやメソッドの定義はタートルから作った「かめた」と同様に行えます。

タグの属性設定

- HTMLタグの属性、プロパティ←CSSには2つの設定方法があります。
- 1つ目は、タグに直接記述する方法です。
    - （例）右寄せを指定したpタグを作成します。

```
段落←<p align="right">右寄せの文章</p>
```

- （例）枠線を持つテーブルを作成します。

```
表←<table border="1"></table>
```

- （例←idを指定したpタグを作成します。

```
段落←<p id="id1">IDを持つ段落</p>
```

- （例←CSSのbackground-colorを指定したspanタグを生成します。

```
テキスト←<span style="background-color: red;">背景が赤色のテキスト</span>
```

- 2つ目は、タグを生成した後にメソッド呼び出しで設定する方法です。

    - （例）idを設定します。

```
段落<p>テキスト</p>
段落"id1" 識別子。
```

- （例）属性の値を取得します。

```
段落<p id="id1"></p>
 <span></span>段落！識別子?）テキスト。
```

- 以下は、さまざまな属性を設定する例です。
    - （例）typeを指定したリストを作成します。

```
リスト<ul type="circle"></ul>
```

- （例）styleを指定して直接スタイルを設定します。

```
掲示板<ul style="border:solid;"></ul>
```

- （例）入力フィールドのタイプを指定します。

```
数値入力<input type="number" value="1"/>
```

- （例）日付入力フィールドを作成します。

```
予約日<input type="date">
```

- （例）チェックボックスを作成します。

```
チェックボックス<input type="checkbox" value="checked">
```

- （例）スライダーを作成します。

```
スライダ<input type="range" list="数値" />。
```

- （例）プレースホルダーを指定します。

```
検索欄<input placeholder="検索ワード"/>。
```

- （例）画像に代替テキスト(alt)を指定します。

```
画像<img src="画像ファイルのURL" alt="画像の説明" class="image">
```

- （例）クラス名を指定します。

```
コンテナ<div class="container" id="container1"></div>
```

- （例）CSSを指定します。

```
テキスト<span>背景が赤色のテキスト</span>
テキスト!"red" 背景色。
```

共通のメソッドと属性

- **テキスト**: タグの内容のテキストを設定します。
  - （例）リスト項目(li)タグにテキストを設定します。

```
メッセージ□<li></li>□□"こんにちは"）テキスト。
```

- **入れる**: divやulのような、他のタグを包括するようなタグに、他のタグをchildNodeとして追加するときに使用します。
  - （例□ulタグにliタグを追加します。

```
メッセージボード□<ul></ul>□
メッセージ□<li></li>□□"こんにちは"）テキスト。
メッセージボード！（メッセージ）入れる。
```

- （例）複数の要素を一度に追加します。

```
表示□<ul></ul>□
要素□□<li>hoge</li>□
表示！（要素１）入れる。
要素□□<li>fuga</li>□
表示！（要素２）入れる。
要素□□<li>piyo</li>□
表示！（要素３）入れる。
```

- **クリア**: 中身をすべて削除するときに使います□innerHTMLごと削除するので□childNodeもinnerTextも消えます。
  - （例□ulタグの内容をクリアします。

```
メッセージボード！クリア。
```

- **動作**: イベントハンドラーで、このプロパティにブロックを代入しておくと、以下のイベントを検出したときに実行します。
  - inputのtypeが[“text”, “search”, “email”, “url”, “number”, “tel”, “password”]の場合は、エンターキーが押されたら
  - inputのそれ以外は値が変更されたら□onchange□
  - それ以外のタグはクリックされたら
  - （例）ボタンがクリックされたときの処理を定義します。

```
送信ボタン□<button>送信</button>□
送信ボタン：動作＝「
  <span></span>□"ボタンが押されました" テキスト。
」□□
```

- （例）入力欄でエンターキーが押されたときの処理を定義します。

```
入力欄□<input type="text">□
入力欄：動作＝「
  <span></span>□□入力欄！値？）テキスト。
」□□
```

- （例）スライダーの値が変更されたときの処理を定義します。

```
スライダ□□<input type="range">□
```

```
結果□<p></p>□
スライダー：動作＝「
 結果！(スライダー！値？) テキスト。
」□□
```

- 動作は「動作設定」メソッドを使用することでも定義することができます。

```
送信ボタン□<button>送信</button>□
送信ボタン！「
  <span></span>□"ボタンが押されました" テキスト。
」 動作設定。
```

idセレクタによる要素の取得

- #id 形式を使ってHTML要素を取得できます。
  - （例□idを指定して要素を取得します。

```
 <span id="output" style="background-color: lightGreen"></span>□
出力欄□#output□
出力欄:テキスト＝"ドリトルを使ってテキストを設定"。
```

フォーム要素のメソッドと属性

- **値？**：フォーム要素の値を取得します□input, textareaなど）。
  - （例）入力欄の値を取得します。

```
入力欄□<input>□
 <span></span>□□入力欄！値？）テキスト。
```

- **値**：フォーム要素の値を設定します。
  - （例）入力欄に値を設定します。

```
入力欄□<input>□
入力欄！（"こんにちは"）値。
```

アラートを表示する

- alert メソッドを使用して、アラートダイアログを表示できます。
  - （例）ボタンをクリックしてアラートを表示します。

```
ボタン□<button>クリック</button>□
ボタン:動作＝「
!("あいうえお") alert□
」□□
```

WebAPIとの連携

- HTMLオブジェクトはWebAPIオブジェクトと組み合わせることで、サーバとの通信を行う双方向コンテンツを作成できます。
- 以下は、簡単なチャットプログラムの例です。

```
メッセージボード=<ul></ul>
入力欄=<input>
送信ボタン=<button>送信</button>

受信_wa = webapi!作る。
受信_wa.url ="read.php"

メッセージ表示＝「｜メッセージ配列｜
 メッセージボード！クリア。
 メッセージ配列！「｜内容｜
  メッセージ=<li></li>=内容）テキスト。
  メッセージボード！（メッセージ）入れる。
」それぞれ実行。
==

メッセージ表示！（受信_wa=読む）実行。

送信_wa = webapi!作る。
送信_wa.url ="write.php"
名前 ="ユーザー"。

送信ボタン：動作＝「
 送信_wa.内容＝名前＋"："＋（入力欄！値？）。
 送信_wa=読む。
 メッセージ表示！（受信_wa=読む）実行。
==
```

- このプログラムでは、サーバ側で動作するPHPプログラム=read.phpとwrite.php=とクライアント側で動作するドリトルプログラムを組み合わせています。
- サーバ側のPHPプログラムは以下のようになります。

snippet.php

```php
//掲示板にメッセージを書き込むwrite.php
<?php
 $message = $_GET['message'] . "\n";
 file_put_contents("chat.txt", $message, FILE_APPEND);
?>

//掲示板の書き込み一覧を取得するread.php
<?php
 $message = file("chat.txt");
 echo json_encode($message);
?>
```

よく使われるHTMLタグ

- ドリトルでは多くのHTMLタグが使用できます。以下はよく使われるタグの例です。
- **ul, li**: 順序なしリストとリスト項目を表示します。
  - （例）順序なしリストを作成します。

```
リスト□<ul></ul>□
項目1□<li></li>□□"項目1"）テキスト。
項目2□<li></li>□□"項目2"）テキスト。
リスト！（項目1）入れる（項目2）入れる。
```

- **ol, li**: 順序ありリストとリスト項目を表示します。
  - （例）順序ありリストを作成します。

```
リスト□<ol></ol>□
項目1□<li></li>□□"項目1"）テキスト。
項目2□<li></li>□□"項目2"）テキスト。
リスト！（項目1）入れる（項目2）入れる。
```

- **input**: 入力欄を表示します。
  - （例）入力欄を作成します。

```
入力欄□<input>□
```

- （例）プレースホルダーを指定した入力欄を作成します。

```
検索入力欄□<input placeholder="検索ワード" id="search-input" />□
```

- （例）数値入力欄を作成します。

```
数量□<input type="number" value="1"/>□
```

- （例）日付入力欄を作成します。

```
予約日□<input type="date">□
```

- **datalist, option**: 入力欄の選択肢リストを作成します。
  - （例）スライダーと選択肢リストを作成します。

```
スライダ□□<input type="range" list="数値" />。
 <datalist id="数値">
  <option value="0"></option>
  <option value="10"></option>
  <option value="20"></option>
  <option value="30"></option>
  <option value="40"></option>
  <option value="50"></option>
 </datalist>□
```

- **textarea**: 複数行の入力欄を表示します。
  - （例）テキストエリアを作成します。

```
内容□<textarea></textarea>□
```

- **button**：ボタンを表示します。
  - （例）ボタンを作成します。

```
ボタン□<button>クリック</button>□
ボタン：動作□□<span></span>□"ボタンがクリックされました" テキスト」。
```

- **iframe**：外部ページをフレーム内に表示します。
  - （例□iframeタグでWebページを表示します。

```
フレーム□<iframe></iframe>□
フレーム□"https://example.com/" ソース。
```

- **div**：汎用的なブロック要素を作成します。
  - （例□divタグを作成します。

```
コンテナ□<div></div>□
テキスト□<p></p>□□"こんにちは"）テキスト。
コンテナ！（テキスト）入れる。
```

- **span**：汎用的なインライン要素を作成します。
  - （例□spanタグを作成します。

```
テキスト□<span></span>□□"こんにちは"）テキスト。
```

- **p**：段落を表示します。
  - （例）段落を作成します。

```
段落□<p></p>□□"これは段落です"）テキスト。
```

- **h1, h2, h3, ...**：見出しを表示します。
  - （例）見出しを作成します。

```
タイトル□<h1>タイトル</h1>□
見出し＝<h2>見出し</h2>□
```

- **img**：画像を表示します。
  - （例）画像を表示します。

```
画像□<img>□
画像□"https://example.com/image.jpg" ソース。
```

- （例）代替テキストとクラスを指定した画像を表示します。

```
画像□<img>□
画像□"example.com/image.jpg" ソース。
画像！"画像の説明" 代替テキスト。
画像□"image" クラス。
```

- **table, tr, th, td**：表を作成します。
  - （例）2行2列の表を作成します。

```
表□<table border="1">
```

```
  <tr>
   <th>工学部</th>
   <th>情報工学部</th>
  </tr>
  <tr>
   <td>電気電子工学科</td>
   <td>情報工学科</td>
  </tr>
</table>
```

- **a** : リンクを作成します。
    - （例）リンクを作成します。

```
リンク<a>リンク</a>
リンク"https://example.com" リンク先。
```

- **br** : 改行を作成します。
    - （例）改行を挿入します。

```
<span>テキスト</span>
<br>
<span>次の行</span>
```

- **hr** : 水平線を作成します。
    - （例）水平線を挿入します。

```
<h1>ドリトルリファレンス</h1>
<hr/>
```

- **label** : フォーム要素のラベルを作成します。
    - （例）ラベルを作成します。

```
<label>ユーザ名:</label>
ユーザ名＝<input type="text"/>
```

スタイル設定

- style要素を使うことでHTMLタグのスタイルをまとめて設定できます。
    - （例）タグ名、id, classを指定してスタイルを設定します。

```
<style>
body{margin: 2%;}
#title{font-size: 30px;}
.submit{background-color: green;}
</style>
```

HTMLオブジェクトの利点

- HTMLタグをプログラム内に直接記述可能なため、動的な画面更新を簡潔に記述できます。

- WebAPIによるサーバ側とクライアント側で処理を分離することで、各処理の責任範囲が明確になり、可読性が高くなります。
- HTMLとプログラムが混在しないため、従来のCGI方式と比較して処理の流れが理解しやすくなります。

実践的な例：メッセージボード

- 以下は、ボタンをクリックするとメッセージを追加表示するシンプルなメッセージボードの例です。

```
メッセージボード□<ul></ul>□
入力欄□<input>□
送信ボタン□<button>追加</button>□

送信ボタン：動作＝「
 メッセージ□<li></li>□□入力欄！値？）テキスト。
 メッセージボード！（メッセージ）入れる。
 入力欄！（""）値。
 □□
```

実践的な例：ショッピングサイト

- 以下は、商品を表示するシンプルなショッピングサイトの例です。

```
 <style> .card-image{width:100px;} </style>□
 <h1>ショッピングサイト</h1>□
 <hr>□
コンテナ = <div></div>□
商品表示=「|商品名 価格 画像URL|
 商品画像=<img class="card-image">□
 商品画像！(画像URL) ソース。
 商品名表示=<h2></h2>□(商品名) テキスト。
 価格表示=<p></p>□("□" + 価格) テキスト。
 コンテナ!(商品画像)(商品名表示)(価格表示)(<hr>)入れる。
 □□

 wa = webapi!作る。
 wa:url = "shopping.php"□
商品リスト = wa!読む。
商品リスト！「|商品|
 商品表示！(商品!1 読む)(商品！2 読む)(商品！3 読む)実行。
」それぞれ実行。
```

- サーバプログラム(shopping.php)

snippet.php

```
<?php
$list = [
 ["coffee", 130, "https://example.com/cofee.jpg"],
```

```
 ["tee", 100, "https://example.com/tee.jpg"]
];
echo(json_encode($list));
```

実践的な例：チャットアプリケーション

- 以下は、合言葉を使ったグループチャットの例です。

```
 <span>あいことば□</span>□
あいことば□<input>□
 <br>□
 <span>名前□</span>□
名前□<input>□
 <br>□
 <span>書き込み内容□ </span>□
書き込み内容□<input>□
 <br>□
送信ボタン□<button>送信</button>□
掲示板□<ul style="border:solid;"></ul>□

送信_wa □ webapi!作る。
送信_wa:url = "chat_send.php"□
送信ボタン：動作＝「
 送信_wa:message□"["+(あいことば:value)+"]"□
 送信_wa:message□送信_wa:message+(名前:value)□"□"□
 送信_wa:message□送信_wa:message+(書き込み内容:value)□
 送信_wa!読む。
 □□

受信_wa □ webapi!作る。
受信_wa:url = "chat_read.php"□
タイマー！作る　３　間隔　６００　時間「
 受信_wa:あいことば＝あいことば:value□
 掲示板:innerHTML□""□
 書き込み一覧＝受信_wa□読む。
 書き込み一覧:data!□□メッセージ｜
  ポスト□<li></li>□
  ポスト:テキスト＝メッセージ。
  ポスト！（掲示板□appendTo□
」それぞれ実行。
」実行。
```

実践的な例：投票サイト

- 以下は、投票機能を持つシンプルな投票サイトの例です。

```
<style>
  .pet-icon{width:100px;}
```

```
  .vote-count{font-weight:bold;}
 </style>
 <h1>あなたはどっち派？</h1>

 <img src="きつね画像のURL" alt="きつねのアイコン" class="pet-icon">
 <h2>きつね派</h2>
きつねButton=<button>投票する</button>
きつね票数<div class="vote-count" id="fox-votes"></div>

 <img src="たぬき画像のURL" alt="たぬきのアイコン" class="pet-icon">
 <h2>たぬき派</h2>
たぬきButton=<button>投票する</button>
たぬき票数<div class="vote-count" id="raccoondog-votes"></div>

 api = webapi!作る。
 api:url = "vote.phpのURL"

結果表示＝「」。
結果表示：作る＝fox raccoondog
 :きつね票数：テキスト=fox+"票"。
 :たぬき票数：テキスト=raccoondog+"票"。


きつねButton:動作=「
  :api:vote="fox"
  r = :api!読む。
  r:results|結果|
 結果表示！(結果!1 読む) (結果!2 読む)作る。
」それぞれ実行。


たぬきButton:動作=「
  :api:vote="raccoondog"
  r = :api!読む。
  r:results|結果|
 結果表示！(結果!1 読む) (結果!2 読む)作る。
」それぞれ実行。


// 初期状態の表示
 r = api!読む。
 r:results|結果|
 結果表示！(結果!1 読む) (結果!2 読む)作る。
」それぞれ実行。
```

実践的な例：ドット絵エディタ

- 以下は、シンプルなドット絵エディタの例です。

```
 <style>
 .pixel{width:6px;height:6px;}
```

```
 .line{display:flex;}
 .canvas{border:solid;}
 </style>

 // WebAPIの設定
 write_wa = webapi!作る。
 write_wa:url = "pixelArt_set.phpのURL"

//色変更とサーバー更新処理
 write = |x y color|
  write_wa:x=x
  write_wa:y=y
  write_wa:c=color
  write_wa!読む。
 

// キャンバスの作成
 canvas = <div class=canvas></div>
 canvas_ary = 配列！作る。
 |y|
  line = <div class=line></div>
  line_ary = 配列！作る。
  |x;pixel|
   pixel = <div class=pixel></div>
   pixel動作＝「
    pixel:element:style:backgroundColor="red"
    write!(x)(y)("red")実行。
   
   pixel!(line)appendTo
   line_ary!(pixel)書く。
」！１００　繰り返す。
  line!(canvas)appendTo.
  canvas_ary!(line_ary)書く。
」！１００　繰り返す。

// サーバーからの状態取得処理
 read_wa = webapi!作る。
 read_wa:url = "pixelArt_get.phpのURL"
タイマー！作る　３　間隔　６００　時間「
  canvas_sv = read_wa!読む。
  canvas_sv   line y
   line   color x; pixel
    pixel = canvas_ary!(y)読む (x)読む。
    pixel:element:style:backgroundColor=color
 」それぞれ実行。
」それぞれ実行。
」実行。
```

実践的な例：予約サイト

- 以下は、日付を指定して予約できるシンプルな予約サイトの例です。

```
<h1>予約サイト</h1>□
result = <div></div>□

reserve = webapi!作る。
reserve:url = "reserve.phpのURL"□
api = webapi!作る。
api:url = "check_availability.phpのURL"□

予約日□<input type="date">□
空き検索□<button>検索</button>□

空き検索:動作=「
  api:date = 予約日:value□
  r = api!読む。
  r:results□□|結果|
 空き状況表示!(結果!1 読む) (結果!2 読む) (結果!3 読む) (結果!4 読む) 作る。
」それぞれ実行。
□□
<br>□

空き状況表示＝「」。
空き状況表示：作る＝□□date time room status□
  sp=<span></span>□
  sp□テキスト□room+"-"+time+"□"□
  □status == "空き"」！なら「
   Button=<button>予約</button>□
   Button:動作=「
    reserve:date = date□
    reserve:time = time□
    reserve:room = room□
    r = reserve!読む。
    □r == "予約が完了しました"」！なら「
     result:テキスト□date+" "+time+" "+room+"の予約が完了しました"。
 」実行。
   □□
」そうでなければ「
   <span>予約済み</span>□
」実行。
  <br>□
自分。
□□
```

実践的な例：ページ切り替え

- 以下は、ボタンをクリックして画面を切り替えるシンプルなページ切り替えの例です。

```
page1 = <div>
<h1>page1</h1>
<p>こんにちは。これはページ１です□</p>
<button id="toPage2">ページ2へ</button>
</div>□
```

```
page2 = <div style="display: none;">
<h1>page2</h1>
<p>こんばんは。これはページ２です</p>
<button id="toPage1">ページ1へ</button>
</div>

#toPage2:動作＝「
 page1:style:display = "none"
 page2:style:display = "block"
」

#toPage1:動作＝「
 page2:style:display = "none"
 page1:style:display = "block"
」
```

実践的な例：ポップアップウィンドウ

- 以下は、ボタンをクリックしてポップアップウィンドウを表示する例です。

```
<style>
.popup-container {
 position: absolute;
 top: 50%;
 left: 50%;
 border: 1px solid;
 transform: translate(-50%, -50%);
}
</style>

popupButton = <button>popup</button>
popupButton:動作 = 「|container|
 container = <div class="popup-container"></div>
 title = <h3>ポップアップウィンドウ</h3>
 closeButton = <button>閉じる</button>
 closeButton:動作＝「
  container! remove
 」
 container!(title) (closeButton) append
」
```